Answer set programs with optional rules: a possibilistic approach*

Kim Bauters[1] and **Steven Schockaert**[2] and **Martine De Cock**[1] and **Dirk Vermeir**[3]

[1] Dept. of Appl. Math., Computer Science and Statistics, Universiteit Gent, Belgium, (kim.bauters, martine.decock)@ugent.be

[2] School of Computer Science & Informatics, Cardiff University, United Kingdom, s.schockaert@cs.cardiff.ac.uk

[3] Department of Computer Science, Vrije Universiteit Brussel, Belgium, dvermeir@vub.ac.be

Abstract

Many problems in artificial intelligence can be encoded as answer set programs (ASP) in which some rules are uncertain. ASP programs with incorrect rules may have erroneous conclusions, but due to the non-monotonic nature of ASP, omitting a correct rule may also lead to errors. To derive the most certain conclusions from an uncertain ASP program, we thus need to consider all situations in which some, none, or all of the least certain rules are omitted. This corresponds to treating some rules as optional and reasoning about which conclusions remain valid regardless of the inclusion of these optional rules. While a version of possibilistic ASP (PASP) based on this view has recently been introduced, no implementation is currently available. In this paper we propose a simulation of the main reasoning tasks in PASP using (disjunctive) ASP programs, allowing us to take advantage of state-of-the-art ASP solvers. Furthermore, we identify how several interesting AI problems can be naturally seen as special cases of the considered reasoning tasks, including cautious abductive reasoning and conformant planning. As such, the proposed simulation enables us to solve instances of the latter problem types that are more general than what current solvers can handle.

1 Introduction

Answer Set Programming (ASP), which is based on the idea of stable models [Gelfond and Lifzchitz, 1988], is a form of non-monotonic declarative programming. The basic idea of ASP is to encode a problem as a set of rules, called a program, such that the models of the program correspond with the solutions of the original problem.

However, many real-world problems are affected by uncertainty, e.g. actions may have uncertain outcomes or we may not have perfect knowledge of the state of a given system. To cope with this, many extensions to ASP have been proposed over the last few years that allow for reasoning over uncertain information [Nicolas *et al.*, 2006; Baral *et al.*, 2009].

As an example, consider the following problem. A firm has a client calling from city C to ask for an appointment the next day. The secretary knows that the sales person is either in city A (inA) or city B (inB), but is more confident that he is in city A. The secretary is almost certain (resp. absolutely certain) that a sales person can get from city A (resp. city B) to city C in one day (toC), assuming there is no road block. There are some rumors of a possible road block on the route from city A to C. Cities A and B also connect to city D, which is definitely reachable in one day from A and B (toD). This problem can be encoded as:

$$\begin{array}{ll} \mathbf{1}: inA \leftarrow not\ inB & \mathbf{0.8}: toC \leftarrow inA, not\ blockAC \\ \mathbf{0.6}: inB \leftarrow not\ inA & \mathbf{1}: toD \leftarrow inA, not\ blockAD \\ \mathbf{0.2}: blockAC \leftarrow & \mathbf{1}: toC \leftarrow inB, not\ blockBC \\ & \mathbf{1}: toD \leftarrow inB, not\ blockBD \end{array}$$

Note that, contrary to classical ASP, we added a weight to each rule to reflect our certainty that the rule is valid. For example, the first two rules on the left express that we are more certain that the sales person is in city A than in city B.

To derive the most certain conclusion, we might choose to omit the least certain rules as in possibilistic logic. Indeed, the rule ($blockAC \leftarrow$) is more likely safe to be excluded since we have a low certainty that the rule is valid, whereas a rule such as ($inA \leftarrow$) may not be omitted. However, if we omit the rule ($blockAC \leftarrow$), then $not\ blockAC$ can be derived in the program, which may allow us to deduce 'toC' when we also know 'inA'. Hence, due to the non-monotonic nature of ASP, both including invalid rules and excluding valid rules may lead to errors. As such, we need to consider all situations in which some, none, or all of the least certain rules are omitted. This naturally leads to the idea of an ASP program with optional rules. Given such a program with optional rules, we want to determine whether particular conclusions hold irrespective of the inclusion of the optional rules.

This idea was recently used in [Bauters *et al.*, 2012] to develop a semantics for Possibilistic ASP (PASP), which we present in detail in Section 2.3. However, this work did not provide implementations, nor does it make explicit the link with programs with optional rules and their applications.

*Kim Bauters is funded by a joint FWO project.

In this paper, we present implementations[1] for the decision problems of ASP programs with optional rules, which we discuss in Section 2.3. Furthermore, we show that the idea of optional rules reaches beyond programs with uncertain rules. In particular, we show how cautious abductive reasoning from logic programs [Eiter *et al.*, 1997] and conformant planning (e.g. [Eiter *et al.*, 2004]) can be seen as special cases of programs with optional rules. In this way, our solver offers the first implementation of cautious abductive reasoning from logic programs, and the first single-step simulation of cautious abductive reasoning in ASP.

The remainder of this paper is organized as follows. In Section 2 we recall some notions from answer set programming and possibility theory, as well as the possibilistic semantics for ASP programs with uncertain rules. In this section, we moreover present the decision problems that will be considered in this paper. In Section 3 we introduce a number of simulations that allow us to reduce the problem of reasoning with uncertain or optional rules to cautious and brave reasoning over classical ASP programs. In Section 4 we explain how cautious abductive reasoning and conformant planning can be reduced to the decision problems of programs with optional rules. In Section 5 we provide an overview of our implementation. Related work is discussed in Section 6 and we formulate our conclusions in Section 7.

2 Preliminaries

2.1 Answer Set Programming (ASP)

A *disjunctive program* is a set of *disjunctive rule*s of the form $l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ with l_i, $0 \leq i \leq n$, a literal, i.e. either an atom 'a' or '$\neg a$' and $k \leq m \leq n$. The operator 'not' denotes negation-as-failure. Intuitively, $not\ l$ is true when we cannot prove l. A disjunctive rule r consists of a head $H(r) = \{l_1, ..., l_k\}$, interpreted as a disjunction, and a body $B(r) = B^+(r) \cup B^-(r)$ with $B^+(r) = \{l_{k+1}, ..., l_m\}$ and $B^-(r) = \{l_{m+1}, ..., l_n\}$, both interpreted as conjunctions. A rule is called a *fact* (resp. *constraint*) when $B(r) = \emptyset$ (resp. $H(r) = \emptyset$). When $B^-(r) = \emptyset$ we say that r is a *positive disjunctive rule*. When $|H(r)| \leq 1$ we say that r is a *normal rule*. A *normal program* is a set of normal rules. $At(P)$ is the set of atoms in program P and $Lit_P = At(P) \cup \{\neg a \mid a \in At(P)\}$.

An *interpretation* I is a set of literals and is *consistent* when there does not exist an atom a with $a \in I$ and $\neg a \in I$. A consistent interpretation I is a *model* of a positive disjunctive rule r when $H(r) \cap I = \emptyset$ or $B^+(r) \not\subseteq I$. For a positive disjunctive program, a consistent interpretation I is a *model* of P when I is a model of every rule in P. An *answer set* of the positive disjunctive program P is a minimal model w.r.t. set inclusion of P. The *reduct* P^I of a disjunctive program P w.r.t. the interpretation I is defined as the set of rules $P^I = \{H(r) \leftarrow B^+(r) \mid r \in P \text{ and } B^-(r) \cap I = \emptyset\}$ [Gelfond and Lifzchitz, 1988]. An interpretation I is an answer set of P when I is an answer set of P^I. P is a *consistent program* when it has at least one answer set. Finally, we write $P \models^{\mathrm{b}} l$ (resp. $P \models^{\mathrm{c}} l$) to denote that l is a brave (resp. cautious) consequence of P, i.e. that an answer set M of P exists with $l \in M$ (resp. that $l \in M$ for all answer sets M of P).

Recall that the complexity classes Σ_n^{P} and Π_n^{P} are defined as $\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$ with $\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$ and $\Pi_{i+1}^{\mathsf{P}} = \mathsf{co}\Sigma_{i+1}^{\mathsf{P}}$, with $\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$ the class of problems solvable in polynomial time on a non-deterministic machine with a Σ_i^{P} oracle, i.e. a procedure that can solve Σ_i^{P} problems in constant time [Papadimitriou, 1994]. Brave reasoning over a disjunctive (resp. normal) program is Σ_2^{P}-complete (resp. NP-complete). Cautious reasoning over a disjunctive (resp. normal) program is Π_2^{P}-complete (resp. coNP-complete) [Baral, 2003].

2.2 Possibility Theory

Possibility theory is a theory for dealing with uncertainty. Let Ω be a (finite) universe and $\omega \in \Omega$. A *possibility distribution* π is defined as a mapping $\pi : \Omega \rightarrow [0, 1]$ and encodes to what extent it is plausible that ω is the actual world. By convention, $\pi(\omega) = 0$ means that ω is impossible and $\pi(\omega) = 1$ means that no available information prevents ω from being the actual world. Possibility degrees are mainly interpreted qualitatively: when $\pi(\omega) > \pi(\omega')$, ω is considered more plausible than ω'. When we impose constraints on a possibility distribution, we are usually only interested in the *least specific possibility distribution*, i.e. the greatest possibility distribution w.r.t. the ordering $>$. A possibility distribution π induces two uncertainty measures [Dubois and Prade, 1988]. The *possibility measure* Π is defined as $\Pi(A) = \max\{\pi(\omega) \mid \omega \in A\}$ with $A \subseteq \Omega$ and evaluates the extent to which a world ω in A is consistent with the beliefs expressed by π. The dual *necessity measure* N is defined as $N(A) = 1 - \Pi(\overline{A})$ and evaluates the extent to which all possible worlds belong to A.

2.3 Possible and Necessary Answer Sets

Possibilistic ASP (PASP) provides a semantics for programs with uncertain rules by interpreting weights associated with rules in terms of a necessity measure. Several proposals have already been made in the literature, including [Nicolas *et al.*, 2006; Bauters *et al.*, 2010]. The following discussion is based on the approach from [Bauters *et al.*, 2012]. A *possibilistic (positive) disjunctive (resp. normal) rule* is a pair (r, c) with r a (positive) disjunctive (resp. normal) rule and $c \in]0, 1]$. A *possibilistic disjunctive (resp. normal) program* is a set of possibilistic disjunctive (resp. normal) rules. For a possibilistic rule $p = (r, c)$ we use p^* to denote r, i.e. the classical rule. Similarly, for a possibilistic program P we use P^* to denote the set of rules $\{p^* \mid p \in P\}$. The set of all weights in a possibilistic program P is denoted $cert(P) = \{c \mid p = (r, c) \in P\}$ and we also use the extended set of weights $cert^+(P) = \{c \mid c \in cert(P)\} \cup \{1 - c \mid c \in cert(P)\} \cup \{0, \frac{1}{2}, 1\}$

Semantically, the weight c is interpreted as the certainty that the rule is valid. For a given program P, all the subprograms $P' \subseteq P$ are considered. Each subprogram P' corresponds with the assumption that omitted rules are wrong and included rules are correct. The possibility of each subprogram P' (in fact, the possibility that the corresponding as-

[1] The prototype of the solver is available online at: `http://www.cwi.ugent.be/kim/pasp2asp/` .

sumption is correct) is determined by looking at the certainties of the rules omitted from P'. Specifically, we write π_P, which is a $\mathcal{P}(P) \rightarrow [0,1]$ mapping, for the least specific possibility distribution satisfying:

1. $\forall (r,c) \in P \cdot N(\{P' \mid P' \subseteq P \text{ and } r \in P'\}) \geq c$;
2. $\pi(P') = 0$ for each inconsistent program P'.

This particular possibility distribution π_P can also be defined over all $P' \subseteq P$ as $\pi_P(P') = 1 - max\{c \mid (r,c) \in P \setminus P'\}$ when P'^* is consistent and $\pi_P(P') = 0$ otherwise.

We now give the main decision problems for ASP with uncertain rules. For each literal the possibility/necessity that the literal is a brave/cautious consequence of P can be determined. Specifically, the degree λ to which 'l' is possibly/necessarily a brave consequence of P is defined as:

$$\Pi\left(P \models^{\mathrm{b}} l\right) = \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{b}} l\right\}$$

$$N\left(P \models^{\mathrm{b}} l\right) = 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{b}} l\right\}$$

The degree to which it is possible/necessary that 'l' is a cautious consequence is defined in a similar way:

$$\Pi\left(P \models^{\mathrm{c}} l\right) = \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{c}} l\right\}$$

$$N\left(P \models^{\mathrm{c}} l\right) = 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{c}} l\right\}$$

Example 1. Consider the example from Section 1, which we name P_{intro}. It is easy to verify that $\Pi\left(P_{intro} \models^{\mathrm{c}} toC\right) \neq 1$. Indeed, it is not possible to guarantee that a sales person will make it in time to city C due to the road block that may prevent travel between cities A and C. However, $\Pi\left(P_{intro} \models^{\mathrm{b}} toC\right) = 1$, i.e. we would not be surprised at all to see that our sales person can make it into city C in one day.

We now give the complexity results. Deciding whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ is NP-complete and coNP-complete, respectively, with P a possibilistic normal program (or Σ_2^{P}-complete and Π_2^{P}-complete, respectively, with P a possibilistic disjunctive program). Deciding whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, however, is Σ_2^{P}-complete and Π_2^{P}-complete with P a possibilistic normal program (or Σ_3^{P}-complete and Π_3^{P}-complete, respectively, with P a possibilistic disjunctive program).

3 Simulation Using ASP

We show how ASP with uncertain rules can be simulated using classical ASP. We start with the simulation of $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, which are the decision problems with the lowest complexity.

Definition 1. Let P be a possibilistic disjunctive program. We define $P_{\mathrm{basic}}(\lambda)$ as the set of rules:

$$\{r' \leftarrow not\ nr' \mid (r,c) \in P, c \leq 1-\lambda\}$$

$$\cup \{nr' \leftarrow not\ r' \mid (r,c) \in P, c \leq 1-\lambda\} \quad (1)$$

$$\cup \{r' \leftarrow \mid (r,c) \in P, c > 1-\lambda\} \quad (2)$$

$$\cup \{head(r) \leftarrow body(r) \cup \{r'\} \mid (r,c) \in P\} \quad (3)$$

Intuitively, the rules (1) generate as many answer sets as there are choices of rules for which the resulting subprogram has a possibility of at least λ. The most certain rules are considered to be valid in (2). The information encoded in the respective rules is applied using (3).

Proposition 1. *Let P be a possibilistic disjunctive program and $P^{\Pi}_{brave}(l,\lambda)$ the disjunctive program defined as $P_{\mathrm{basic}}(\lambda) \cup \{\leftarrow not\ l\}$. Then $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ iff $P^{\Pi}_{brave}(l,\lambda)$ has a classical answer set.*

Proof. The condition $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is equivalent to $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{b}} l\right\} \geq \lambda$ which states that there is some subprogram $P' \subseteq P$ with $P'^* \models^{\mathrm{b}} l$ such that $\pi_P(P') \geq \lambda$. The latter condition is equivalent to $P_{\mathrm{req}} \subseteq P'^*$ with $P_{\mathrm{req}} = \{r \mid (r,c) \in P, c > 1-\lambda\}$. Thus the problem reduces to determining whether for some set of rules $P_{\mathrm{opt}} \subseteq \{r \mid (r,c) \in P, c \leq 1-\lambda\}$ we have $(P_{\mathrm{req}} \cup P_{\mathrm{opt}}) \models^{\mathrm{b}} l$. By construction of $P_{\mathrm{basic}}(\lambda)$, due to the rules (2), we know that every rule in P_{req} is chosen. Furthermore, every choice made in (1) corresponds with a choice of P_{opt}. Finally, the addition of the rule $\{\leftarrow not\ l\}$ ensures that 'l' must be a conclusion of some answer set of the simulation $P^{\Pi}_{\mathrm{brave}}(l,\lambda)$, or otherwise $P^{\Pi}_{\mathrm{brave}}(l,\lambda)$ will not have any answer sets. □

Proposition 2. *Let P be a possibilistic disjunctive program, $\lambda > 0$ and $P^{N}_{cautious}(l,\lambda)$ the disjunctive program defined as $P_{\mathrm{basic}}(1-\lambda') \cup \{\leftarrow l\}$ with $\lambda' \in cert^{+}(P)$ such that $\lambda' < \lambda$ and $\nexists \lambda'' \in cert^{+}(P) \cdot \lambda' < \lambda'' < \lambda$. Then $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $P^{N}_{cautious}(l,\lambda)$ has no classical answer set.*

Proof. The proof is analogous to the proof of Proposition 1. To determine whether $N(P \models^{\mathrm{c}} l) \geq \lambda$ we need to verify that $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{c}} l\right\} \leq 1-\lambda$. In other words, we need to verify that there does not exist a subprogram P' such that $P'^* \not\models^{\mathrm{c}} l$ and $\pi_P(P') > 1-\lambda$. The simulation $P^{N}_{cautious}(l,\lambda)$ looks for a subprogram with a certainty strictly higher than $1-\lambda$ in which 'l' is false, i.e. l is not a cautious consequence. Furthermore, note that the certainty will be strictly higher than $1-\lambda$ iff it is at least $1-\lambda'$. If this does not exist, i.e. if we find no answer sets, then $N(P \models^{\mathrm{c}} l) \geq \lambda$. □

For the decision problems in Proposition 1 and 2, it was sufficient to find *one answer set* of a particular subprogram that satisfies some condition, which is why we were able to simulate these problems relatively straightforwardly. To decide whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ or $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, on the other hand, we need to verify a particular condition *for each answer set* of a particular subprogram. Since the complexity of these decision problems is higher, we are only able to provide simulations for P a possibilistic normal program as these are already Σ_2^{P}-hard and Π_2^{P}-hard, respectively.

Our simulation is based on the idea that we can use a disjunctive ASP program to reason about the answer sets of a normal ASP program. This will be accomplished by translating the normal ASP program to a set of clauses to ensure that the program is free of negation-as-failure. This is needed to be able to apply saturation techniques over a normal program.

If a program is tight, then a translation to clauses can be obtained by determining the completion of the original program [Fages, 1994]. To define tightness, we first need to define the *dependency graph* of an ASP program. This graph is the directed graph with signed edges (either $+$ or $-$) such that vertices of the graph are the literals mentioned in P [Baral,

2003]. There is a *directed positive edge* from l_i to l_0 if there is a rule $r \in P$ such that $H(r) = l_0$ and $l_i \in B^+(r)$. Similarly, there is a *directed negative edge* from l_i to l_0 if there is a rule $r \in P$ such that $H(r) = l_0$ and $l_i \in B^-(r)$. There is a *positive path* from l_1 to l_2 iff there is a path in the dependency graph from vertex l_1 to vertex l_2 consisting only of positive edges. A normal program P is said to be *tight* if it does not contain a *positive cycle*, i.e. a positive path starting and ending in a vertex l [Lin and Zhao, 2003].

Not every ASP program, however, is tight. As such, we will need to rely on more complex translations of ASP programs to sets of clauses, such as the translation based on a characterization in terms of level numbering presented in [Janhunen, 2004]. Once we have the translation to a set of clauses, we generate answer sets for every subprogram and we apply saturation techniques to both validate whether a given interpretation is a valid model of the subprogram and to verify whether a given literal is a desired conclusion of the given subprogram.

Definition 2. Let P be a possibilistic normal program. The disjunctive program $P_{complex}(\lambda)$ is defined as the set of rules:

$$\{\mathrm{r}' \leftarrow not\ \neg\mathrm{r}' \mid (r,c) \in P, c \leq 1-\lambda\}$$
$$\cup \{\neg\mathrm{r}' \leftarrow not\ \mathrm{r}' \mid (r,c) \in P, c \leq 1-\lambda\} \quad (4)$$
$$\cup \{\mathrm{r}' \leftarrow \mid (r,c) \in P, c > 1-\lambda\} \quad (5)$$

$$\cup \left\{cl \leftarrow \mid cl \in cls(P^r)^\dagger\right\} \quad (6)$$
$$\cup \left\{(sat \leftarrow a, na) \mid a \in at(cls(P^r)^\dagger)\right\} \quad (7)$$
$$\cup \left\{(a \leftarrow sat) \mid a \in at(cls(P^r)^\dagger)\right\}$$
$$\cup \left\{(na \leftarrow sat) \mid a \in at(cls(P^r)^\dagger)\right\} \quad (8)$$
$$\cup \{\leftarrow not\ sat\} \quad (9)$$

$$\cup \left\{cl'_r \leftarrow \mid cl \in cls(P^r)^\dagger\right\} \quad (10)$$
$$\cup \left\{\leftarrow a', na' \mid a \in at(cls(P^r)^\dagger)\right\} \quad (11)$$

where $cls(P)$ is a representation of the normal program P as a set of clauses (e.g. [Janhunen, 2004]), P^{r} is the set of rules $\{head(r) \leftarrow body(r), \mathrm{r} \mid r \in P\}$ with 'r' a fresh atom, $C^\dagger$ is the set of clauses obtained from C by replacing every occurrence of a negated atom $\neg a$ with a fresh atom na except for the atoms r_i and $at(C)$ is the set of atoms appearing in C from which we remove the atoms r_i. Finally, cl'_r is obtained from a clause cl by replacing every literal from Lit_P with l'.

Proposition 3. *Let P be a possibilistic normal program and $P^{\mathrm{c}}_{\Pi}(l,\lambda)$ the disjunctive program defined as $P_{complex}(\lambda) \cup \{sat \leftarrow l\}$. Then $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $P^{\mathrm{c}}_{\Pi}(l,\lambda)$ has a classical answer set.*

Proof. We want to determine whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, i.e. whether there exists a $P' \subseteq P$ such that $P'^* \models^{\mathrm{c}} l$ and $\pi_P(P') \geq \lambda$. The latter condition means that $(r,c) \in P'$ for every $(r,c) \in P$ with $c > 1-\lambda$. Similar as in Proposition 1, the rules in (4) and (5) generate as many answer sets as there are subprograms $P' \subseteq P$ for which $\pi_P(P') \geq \lambda$.

For each such subprogram P' we want to determine whether P'^* has 'l' as a cautious conclusion. By construction, $\{cl \leftarrow \mid cl \in cls(P^r)\}$ is equivalent to P^r. In particular, every model of these rules corresponds to an answer set of P^r. Since we removed classical negation in (6), however, we need to add the rules in (7) to ensure that 'sat' is contained in the answer set whenever 'a' and the opposite atom 'na' are true at the same time. The intuition of making 'sat' true is thus to indicate that this is not a valid answer set of the subprogram P'. The rule $(l \leftarrow sat)$ is used to try to make 'l' false, by once again ensuring that 'sat' is contained in the answer set whenever 'l' is in the answer set. Intuitively, we thus say that an answer set in which 'l' is true is undesirable, i.e. we prefer answer sets of P' in which 'l' is false. The rule (9) is then used to block all answer sets in which 'sat' is false. In other words: unless for every answer set of P' we have that 'l' is true in the answer set, we have not found that 'l' is a cautious consequence of P'.

Thus far we have not discussed the use of the rules (8). Together with the atom 'sat', these rules are used to implement a saturation technique [Baral, 2003] over our disjunctive simulation and we refer to this work for a detailed overview of how saturation works. The intuition of saturation is that we use the property that an answer set is a *minimal* model. In particular, the rules in (8) will add all the atoms under consideration to the model M to try and prevent it from being an answer set. Indeed, if we find a model $M' \subseteq M$ then clearly M cannot be an answer set. As such, we can ensure that consistent models of P' are preferred over inconsistent models, and that models of P' in which 'l' is false are preferred over models in which 'l' is true. Then, only if no consistent answer set (in which 'l' is false) exists for P', will we have that 'sat' is true in an answer set of $P^{\mathrm{c}}_{\Pi}(l,\lambda)$.

Finally, when a subprogram P' is inconsistent, then $\pi(P') = 0$, i.e. we do not want to consider this subprogram. Notice, however, that the rule (7) would not work as expected in this case. Indeed, if P' is inconsistent it does not have a consistent model and the saturation technique would not exclude this subprogram. As such, we repeat our simulation of the subprogram P' in (10) and use constraints in (11) to effectively block inconsistent subprograms. □

Proposition 4. *Let P be a possibilistic normal program and $P^{\mathrm{b}}_{\mathrm{N}}(l,\lambda)$ the disjunctive program defined as $P_{complex}(1-\lambda') \cup \{sat \leftarrow not\ l\}$ with λ' defined as in Proposition 2. Then $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ iff $P^{\mathrm{b}}_{\mathrm{N}}(l,\lambda)$ has no classical answer set.*

Proof. This proof is analogous to the proof of Proposition 3, similar as how the proof of Proposition 2 was analogous to the proof of Proposition 1. □

4 Certain programs with optional rules

Our simulations for the decision problems $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ are not only useful for reasoning with uncertain answer set programs, but can be applied to the much wider problem range of programs with optional rules. In particular, in this section we prove how two interesting AI problems, namely cautious abductive reasoning and conformant planning, can be expressed in terms of programs with optional rules. As such, we can solve these problems with the

simulation offered in Section 3 and this, in turn, forms the first implementation of cautious abductive reasoning and the first single-step implementation of conformant planning in ASP. Both problems can also trivially be extended with weights.

4.1 Cautious abductive reasoning

An abductive diagnosis program [Eiter *et al.*, 1997] is encoded as a triple $\langle H, T, O\rangle$ where H is a set of literals referred to as hypotheses, T is a (normal) ASP program referred to as the theory and O is a set of literals referred to as observations. Intuitively, the theory T describes the dynamics of a system, the observations O describe the observed state of the system and the hypotheses H are those literals that can be used to try and explain such observations within the theory. Cautious abductive reasoning is concerned with the problem of finding hypotheses that could explain the observations in O. Thus, we are interested in a set $E \subseteq H$ such that $T \cup E \models^{\mathrm{c}} O$, where E is said to be a cautious explanation.

Proposition 5. *Let P_{abd} be the possibilistic normal program defined for an abductive diagnosis program $\langle H, T, O\rangle$ as*

$$\{\mathbf{0.5}{:}\mathit{block\_h} \leftarrow \mid h \in H\} \tag{12}$$
$$\cup\, \{\mathbf{1}{:}h \leftarrow \mathit{not}\ \mathit{block\_h} \mid h \in H\} \tag{13}$$
$$\cup\, \{\mathbf{1}{:}\mathit{goal} \leftarrow O\} \tag{14}$$
$$\cup\, \{\mathbf{1}{:}r \mid r \in T\}\,. \tag{15}$$

It holds that $\langle H, T, O\rangle$ has a cautious explanation iff $\Pi\left(P_{\mathrm{abd}} \models^{\mathrm{c}} \mathit{goal}\right) \geq 0.5$. In particular, E is a cautious explanation iff for $P' = P_{\mathrm{abd}} \setminus \{\mathit{block\_h} \leftarrow \mid h \in E\}$ we have that $P' \models^{\mathrm{c}} \mathit{goal}$.

Proof. For $\Pi\left(P_{\mathrm{abd}} \models^{\mathrm{c}} \mathit{goal}\right) \geq 0.5$, we must have some $P' \subseteq P_{\mathrm{abd}}$ such that $P' \models^{\mathrm{c}} \mathit{goal}$ and $\pi(P') \geq 0.5$. Thus, clearly, all the rules defined in (13), (14) and (15) must be in P'. It is furthermore easy to see that for every $h \in H \setminus E$ we have that $(h \leftarrow) \in \left((P')^*\right)^M$ for every answer set M of P'^* and, since $P'^* \models^{\mathrm{c}} \mathit{goal}$, that E is a cautious explanation. □

Example 2. John wants to become rich. He can either choose to invest his money in stocks, or to invest it in bonds (he lacks the money to do both). He can either win or fail with his investment, where he resp. becomes rich or bankrupt. Bonds are safer and will make him rich, eventually. Whether or not he should invest will also depend on how certain he is that his investment will succeed (very certain) or fail (somewhat certain) and his confidence that investing in bonds will make him rich (somewhat certain):

$$\mathbf{0.8}{:}\mathit{win} \leftarrow \mathit{not}\ \mathit{fail}, \mathit{invest} \qquad \mathbf{0.5}{:}\mathit{fail} \leftarrow \mathit{not}\ \mathit{win}, \mathit{invest}$$
$$\mathbf{1}{:}\mathit{rich} \leftarrow \mathit{win} \qquad \mathbf{1}{:}\mathit{bankrupt} \leftarrow \mathit{fail}$$
$$\mathbf{0.5}{:}\mathit{rich} \leftarrow \mathit{bonds} \qquad \mathbf{1}{:}\leftarrow \mathit{invest}, \mathit{bonds}$$

Given the hypotheses $H = \{\mathit{invest}, \mathit{bonds}\}$, it is clear that when we ignore the weights only $E = \{\mathit{bonds}\}$ is a cautious abductive explanation for the observation $O = \{\mathit{rich}\}$. If we take the certainties into account and $\lambda = 0.5$, then both $E_1 = \{\mathit{bonds}\}$ and $E_2 = \{\mathit{invest}\}$ are cautious explanations.

4.2 Conformant planning

Conformant planning is the problem of determining whether a plan (i.e. a series of actions) exists that always leads to the desired goal, regardless of the incompletely known initial state of the agent. Such problems are typically expressed using an action language.

An action language is built from a finite number of *fluent*s $f_1, ..., f_n$. A *state* is a finite set of fluents. The properties of the *initial state* s_0 are described by formulas of the form 'initially f', which are called *value propositions*, with f a *fluent literal*, i.e. a fluent or a fluent preceded by $\neg$. Changes of states are defined using a finite number of *action*s $a_1, ..., a_j$. Formulas of the form 'a causes f if $f_1, ..., f_m$' are called *effect propositions*, with $f, f_1, ..., f_m$ fluent literals and a an action. A *domain* D is a finite set of value and effect propositions. A *proper* domain, to which we limit ourselves in this paper, is a domain in which we can determine in polynomial time what the successor state is, given the current state and an action. A *plan* is a sequence of actions $[a_1, ..., a_m]$. The *planning problem* is to determine for a given domain D and a fluent literal f whether a plan exists leading from s_0 to a state in which f is true, where we call f the *goal fluent*. To solve a planning problem, the domain is translated to ASP. Particularly, such a translation can be written as $P_{\mathrm{act}} \cup P_{\mathrm{rem}}$ where P_{act} are those rules used to describe the actions, whereas P_{rem} are the remaining rules that among others describe the (incomplete) initial state and rules to ensure inertia. Then, a plan exists when an answer set contains the goal fluent.

However, not all forms of planning problems can be solved in this way. When we say that we have an incomplete domain, this means that the initial values of some fluents are unknown. *Conformant planning* is the problem of determining whether for an incomplete domain and a fluent f a plan exists leading to a state in which f is true, regardless of the initial values of the unknown fluents. Only some action languages, e.g. $\mathcal{K}$ [Eiter *et al.*, 2000; Eiter *et al.*, 2004], have the expressive power to describe conformant planning problems. For solving such problems, $DLV^{\mathcal{K}}$ relies on a two-step translation to ASP where a plan is generated (that is not necessarily a conformant plan) and verified to be an actual conformant plan, until an actual conformant plan is found. However, these methods are not designed to work with uncertainty and cannot, e.g. compute the most reliable plan when no conformant plan can be found.

We now show how conformant planning can be expressed in terms of a decision problem of the form $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Note that the existence of a conformant plan can be also written as $\exists p\, \forall iv \cdot P(p, iv, pp)$ where $P(p, iv, pp)$ describes that for the planning problem pp and for all initially unknown values iv the plan p leads to the goal fluent.

Proposition 6. *Let P_{con} be the possibilistic normal program defined for a conformant planning problem with the atom 'goal' the desired goal fluent. We express the domain knowledge as a normal ASP program $P_{\mathrm{act}} \cup P_{\mathrm{rem}}$. Then P_{con} is:*

$$\{\mathbf{0.5}{:}\mathit{block\_i} \leftarrow \mid r_i \in P_{\mathrm{act}}\} \tag{16}$$
$$\cup\, \{\mathbf{1}{:}H(r_i) \leftarrow B(r_i) \cup \{\mathit{not}\ \mathit{block\_i}\} \mid r_i \in P_{\mathrm{act}}\} \tag{17}$$
$$\cup\, \{\mathbf{1}{:}r \mid r \in P_{\mathrm{rem}}\} \tag{18}$$

$$\cup \{\mathbf{1}: \leftarrow not\ goal\} \tag{19}$$

A conformant plan exists iff $\Pi\,(P_{\text{con}} \models^{\text{c}} goal) \geq 0.5$.

Proof. When $\Pi\,(P_{\text{con}} \models^{\text{c}} goal) \geq 0.5$ then, by definition, there exists a subprogram $P' \subseteq P$ such that $(P')^* \models^{\text{c}} goal$ with $\pi_P(P') \geq 0.5$. Since $\pi_P(P') \geq 0.5$ we know that all the rules from (17), (18) and (19) are in P'. Thus, only rules from (16) may be in $P \setminus P'$. In that case, the corresponding rule from (17) ensures that for every answer set M of $(P')^*$ we have that $(H(r_i) \leftarrow B(r_i)) \in ((P')^*)^M$. Thus, the action is no longer blocked and can be applied. Because of the available actions we can, regardless of the initial state described in (18), cautiously derive '$goal$'. Indeed, otherwise we know due to (19) that M is not be a model. In other words: the choice made in (16) corresponds with a set of actions that form a cautious plan for the given planning problem. □

Example 3. Consider the example from the introduction where we add additional action rules (e.g. $driveAtoC$). Clearly, by adding certainties, we can compute a conformant plan with a lower certainty, even when a classical conformant plan (i.e. with absolute certainty) does not exist.

5 Implementation

We have implemented an operational prototype of the simulation presented in Section 3. The translator PASP2SAT prepares an input PASP program by removing the certainties and adding the fresh atom r_i to the body of each rule $r_i \in P$. The resulting ASP program is converted to an equivalent set of clauses using the technique from [Janhunen, 2004]. The output in DIMACS CNF form is converted back into ASP rules by CLAUSE2ASP. The rules (6) – (11) are constructed from the information of the translation to clauses, while λ is needed to add the rules (4) – (5) and l is required to add the rules that are specific to the simulation of the decision problem as identified in Proposition 3 and 4. An overview of the implementation is given in Figure 1. The answer sets of the resulting ASP program can then be computed using an ASP solver for disjunctive programs, e.g. DLV [Eiter *et al.*, 1999; Leone *et al.*, 2006].

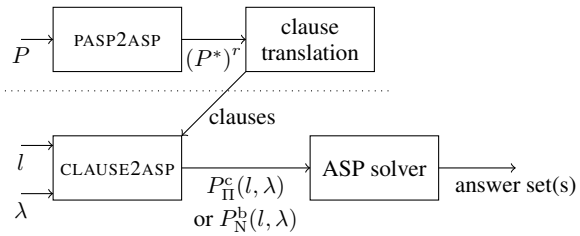

Figure 1: overview of the implementation.

We compared our solver on the conformant planning problem against $DLV^{\mathcal{K}}$, where both solvers use a world-state encoding in action language $\mathcal{K}$ of the $BMTUC(p,t)$ problem[2] [Eiter *et al.*, 2003].

[2]Bomb in Toilet problem with uncertain clogging, concurrent dunks, t toilets and p packages.

For $BMTUC(2,4)$, $BMTUC(3,4)$ and $BMTUC(4,4)$ both our solver and $DLV^{\mathcal{K}}$ were able to find a secure plan in less than 1 second. For $BMTUC(5,4)$, within one second, a secure plan could still be found by $DLV^{\mathcal{K}}$ but not by our solver. For $BMTUC(6,4)$ neither of the solvers was able to find a solution within 1 second. For other variants of the Bomb in Toilet problem, a similar pattern was observed. Although our solver is somewhat slower than $DLV^{\mathcal{K}}$, it is competitive on most problem instances. Moreover, while $DLV^{\mathcal{K}}$ is optimized for the problem of conformant planning, our solver is more generic and can thus not exploit problem-specific heuristics. Moreover, in these experiments we did not consider optimization, based on the tightness of the programs.

6 Related Work

The combination of uncertainty with logic programming has been widely studied. One the earliest results on how to combine probability theory and logic programming is [Lukasiewicz, 2002]. Later, in [Baral *et al.*, 2009], a framework was proposed in which ASP is combined with probability theory. In this framework, probabilistic atoms are used in addition to classical ASP to describe the probability that the atom will take on a random value given the prior knowledge.

The work in [Dubois *et al.*, 1994] on possibilistic logic is one of the first approaches to combine possibility theory with logic programming. The first work on combining ASP and possibility theory was [Nicolas *et al.*, 2006], which introduced the PASP framework. In this approach, '$not\ l$' is treated as *it is more certain that '*$\neg l$*' holds*. In [Bauters *et al.*, 2010], an alternative was presented, where '$not\ l$' is interpreted as *the degree to which it is possible that '*$\neg l$*'*. Essentially, both approaches can be seen as a multi-valued logic, using the negation from Gödel logic and Łukasiewicz logic, respectively. These two approaches have in common that the weights associated with rules are used to obtain weighted answer sets. The approach from [Bauters *et al.*, 2012], however, which we have adopted in this paper, uses the weights to obtain a weighted set of classical answer sets. As such, a program can be seen as a set of uncertain rules.

To the best of our knowledge, the approach presented in this paper is the first implementation of cautious abductive reasoning. Brave abductive reasoning, on the other hand, has seen a myriad of implementations and many solvers for answer set programming, e.g. [Leone *et al.*, 2006; Gebser *et al.*, 2011], have incorporated very performant mechanisms for brave abductive reasoning. Interestingly, [Dubois and Prade, 1995] illustrated how abductive reasoning can benefit from possibility theory, allowing to order the possible cautious explanations. Also of interest is the work from [Medinaús *et al.*, 2001] on abduction in multi-adjoint logic programs. In multi-adjoint logic programs, it is possible to associate confidence factors with the rules. Furthermore, it allows to specify for each rule the type of the implication, i.e. Gödel, Łukasiewicz or the product. Still, no practical implementation has been suggested for this particular type of abduction.

Conformant planning, which is also known under other names such as secure planning and strong planning, has also

seen a lot of interest. For a fixed plan length, conformant planning is a Σ_3^{P}-complete problem and secure checking, i.e. verifying whether a plan is a secure plan, is a Π_2^{P}-complete problem. If we restrict ourselves to proper planning domains, then conformant planning and secure checking is Σ_2^{P}-complete and coNP-complete, respectively. Many implementations of conformant planning exist, including $\mathcal{C}$-Plan [Ferraris and Giunchiglia, 2000], CMBT [Cimatti and Roveri, 2004], Conformant-FF [Hoffmann and Brafman, 2006], and DLV$^{\mathcal{K}}$ [Eiter *et al.*, 2004], where the latter is an ASP-based approach. The latter approach, in particular, is an ASP-based approach in which the planning problem is expressed in the action language $\mathcal{K}$. Contrary to our approach, however, these implementations do not allow certainties.

7 Conclusions

We have considered the problem of reasoning with uncertain answer set programs and have provided the first implementations for each of the main reasoning tasks. Taking advantage of recent progress on efficient translations from ASP to SAT, our approach translates an answer set program with uncertain rules into a classical (possibly disjunctive) answer set program, such that the answer sets of the latter program correspond to solutions of the original problem. We showed the practical significance of our implementation, beyond managing uncertainty in ASP, by showing how cautious abductive reasoning and conformant planning can be naturally seen as special cases of the considered problem. To the best of our knowledge, our approach is the first implementation of cautious abductive reasoning from ASP programs and the first implementation of conformant planning that is exclusively based on a translation to a single classical ASP program.

References

[Baral *et al.*, 2009] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009.

[Baral, 2003] Chitta Baral. *Knowledge, Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[Bauters *et al.*, 2010] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir. Possibilistic answer set programming revisited. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI'10)*, 2010.

[Bauters *et al.*, 2012] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir. Possible and necessary answer sets of possibilistic answer set programs. In *Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 836–843, 2012.

[Cimatti and Roveri, 2004] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1–2):127–206, 2004.

[Dubois and Prade, 1988] Didier Dubois and Henry Prade. *Possibility theory: an approach to computerized processing of uncertainty*. Plenum Press, 1988.

[Dubois and Prade, 1995] Didier Dubois and Henri Prade. Fuzzy relation equations and causal reasoning. *Fuzzy Sets and Systems*, 75(2):119–134, 1995.

[Dubois *et al.*, 1994] Didier Dubois, Jérôme Lang, and Henry Prade. Possibilistic logic. *Handbook of Logic for Artificial Intelligence and Logic Programming*, 3(1):439–513, 1994.

[Eiter *et al.*, 1997] Thomas Eiter, Georg Gottlob, and Nicola Leone. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1–2):129–177, 1997.

[Eiter *et al.*, 1999] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communication*, 12(1-2):99–111, 1999.

[Eiter *et al.*, 2000] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Planning under incomplete knowledge. In *Proceedings of the First International Conference on Computational Logic (CL 2000)*, pages 807–821, 2000.

[Eiter *et al.*, 2003] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, II: The $DLV^{\mathcal{K}}$ system. *Artificial Intelligence*, 144(1–2):157–211, 2003.

[Eiter *et al.*, 2004] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Transactions on Computational Logic*, 5(2):206–263, 2004.

[Fages, 1994] François Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1(1):51–60, 1994.

[Ferraris and Giunchiglia, 2000] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 748–753, 2000.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Communications*, 24:107–124, 2011.

[Gelfond and Lifzchitz, 1988] Michael Gelfond and Vladimir Lifzchitz. The stable model semantics for logic programming. In *Proceedings of the 5th Joint International Conference and Symposium on Logic Programming (ICLP'88)*, pages 1081–1086, 1988.

[Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[Janhunen, 2004] Tomi Janhunen. Representing normal programs with clauses. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'04)*, pages 358–362, 2004.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[Lin and Zhao, 2003] Fangzhen Lin and Jicheng Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI'03)*, pages 853–858, 2003.

[Lukasiewicz, 2002] Thomas Lukasiewicz. Probabilistic default reasoning with conditional constraints. *Annals of Mathematics and Artificial Intelligence*, 34(1–3):35–88, 2002.

[Medinaús *et al.*, 2001] Jes Medinaús, Ojeda-Manuel Aciego, and VojtáPeter s. A multi-adjoint logic approach to abductive reasoning. In *Proceedings of the 17th International Conference on Logic Programming (ICLP'01)*, pages 269–283, 2001.

[Nicolas *et al.*, 2006] Pascal Nicolas, Laurent Garcia, Igor Stéphan, and Claire Lefèvre. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47(1–2):139–181, 2006.

[Papadimitriou, 1994] Christos Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.